

Flexible Checker: A one-stop shop for all your checkers and a methodology for elastic scoreboarding

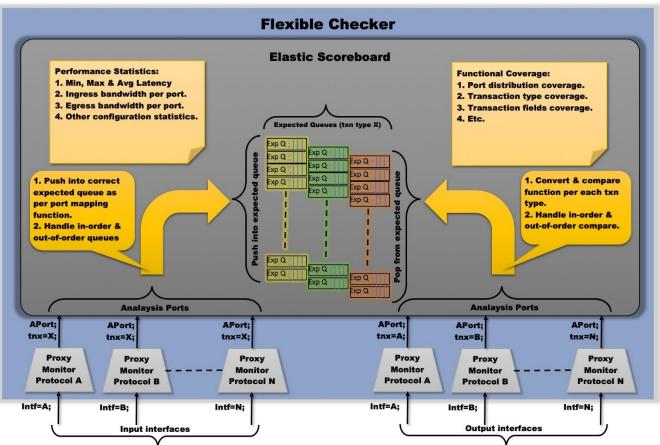
Saad Zahid, Chandra Veedhi, Sumit Dhamanwala Arteris Inc 9601 Amberglen Blvd. Building G, Suite 117, Austin TX, 78729

Abstract—This paper aims to define a methodology for a composite checker/scoreboard called Flexible Checker that provides test-bench writers a solution that can adapt to ever-changing design configurations. Flexible Checker also provides a solution that is portable across blocks, sub-blocks, and chip-level environments. It can also support flexible "transaction trace checking," which checks transactions at multiple internal nodes before reaching their end points. Flexible Checker is equipped to record and dump vital statistics, including bandwidth, latency, traffic distribution, transaction count (type, size, etc.), from end to end in the report phase of a UVM.

I. INTRODUCTION

The scoreboard for the UVM test bench of a SoC is a crucial piece that checks the data integrity across the DUT (design under test). The way a scoreboard is coded traditionally varied from engineer to engineer, configuration to configuration, and block to block. In this paper, we will show how to templatize a scoreboard to make it more configurable, scalable, and lightweight at the same time. To achieve this, we introduce what we call a **proxy transaction container class**, a **proxy monitor**, and **proxy transaction queues**.





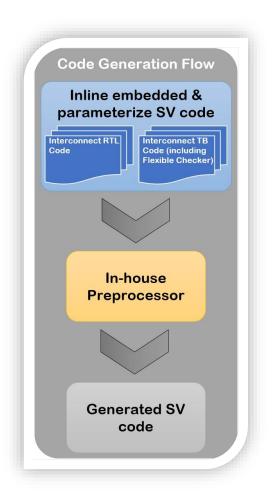
II. ARTERIS VERIFICATION CHALLENGE

For companies like Arteris, which provides IP interconnects with the capability to be constituted of an infinite number of configurations, it is essential to make the corresponding test benches fully flexible and elastic yet robust and effective to deliver high quality. Our biggest challenges to consider during the development process are as follows:

- 1. Each and every interface field and position is configurable.
- 2. Each block/unit/component is also fully elastic in terms of
 - a. The number of input/outputs,
 - b. how its features can be enabled/disabled from a code-generation prospective, and
 - c. the buffer sizes, pipeline stages, states, registers, etc.

3. Similarly, all the above must be complemented by an equivalent test bench to verify each available feature accordingly. These above-mentioned challenges cannot be solved just by using the SystemVerilog (SV) parameter alone, so we have to develop a custom tool to preprocess our embedded (inline) and parameterized code to generates the final SV code.





III. KEY COMPONENTS

To make Flexible Checker highly scalable, agile, and portable, we needed to make a few fundamental verification strategy decisions and develop the key components as follows:

- 1. All in-house UVCs (or VIPs) must have a portable monitor so that we can instantiate it directly in Flexible Checker.
- 2. For external (third-party) VIPs, we wrote a proxy monitor to collect third-party transactions and embed them into our proxy transaction container class and pass them to an elastic scoreboard.
- 3. All test-bench components must use a single transaction type (proxy transaction container) regardless of the actual transaction type (for example, AXI, APB, AHB, or any number of in-house protocols).
- 4. We assembled a wrapper class (extended from UVM_COMPONENT) called Flexible Checker, which includes all the required monitors and the elastic scoreboard.

IV. PROXY TRANSACTION CONTAINER CLASSES

The proxy transaction item class (3rdparty_txn_item) is both a container class for porting a third-party VIP as well as a base class for an in-house VIP. It is the transaction class from which other protocol-specific transaction items are derived. It also has a handle defined as the uvm_sequence_item type, which is used as a container for third-party VIP transaction items. It defines transaction query operations such as "get destination," "get source," and "get ID" as virtual functions, which are supposed to be overloaded in the extended classes. The code is shown below.



18 `uvm_fatal(get_name(), "base get_addr() called") 26 extvip.seq_item e_item; 19 endfunction: get_addr 36 20 'urtual function int get_source(); 37 return e_item.addr; 21 'urtual function int get_source(); 31 'urtual function int get_source(); 22 `uvm_fatal(get_name(), "base get_source() called") 36 \$cast(e_item, twn_handle); 23 endfunction: get_source 36 scast(e_item, twn_handle); 24 'urtual function int get_dest(); 'urtual function int get_dest(); itrus function int get_dest(); 26 'urtual function int get_dest(); 'urtual function int get_dest(); 'urtual function int get_dest(); 27 endfunction: get_destination 'urtual function void my_compare(flexible_seq_item txn); 'scast(e_item, twn_handle); 28 virtual function void my_compare(flexible_seq_item txn); 'urus function void my_compare(flexible_seq_item txn); 'urus function void my_compare(flexible_seq_item txn); 27 endfunction: my_compare 'if(_item.get_data() != twn.get_data())`uvm_error(get_name(), "Data mism if(e_item.get_data() != twn.get_dat())`uvm_error(get_name(), "Address m 28 'endif //FLEXIBLE_SEQ_ITEM 'iendif //Srdpartyyip_seq_item	U1	UNITED STATES						
<pre>infinderFLEXIBLE_SEQ_ITEM idefineFLEXIBLE_SEQ_ITEM idefineFLEXIBLE_SEQ_ITEM</pre>	flexible_seq_item.sv			3rdPartyVIP_Seq_Item.sv				
<pre>2 'defineFLEXIBLE_SEQ_ITEM</pre>		`ifndef FLEXIBLE SE0 ITEM	1					
<pre>class flexible_seq_item extends uvm_sequence_item; uvm_sequence_item txn_handle; // citem txn_handle; // note tn_handle medis to be assigned to the extip_seq_item // uvm_sequence_item // Listing all the virtual function to be extended in derived // UVC sequence item virtual function new(string name); super.new(name); endfunction: new virtual function bit (#095:0] get_data(); virtual function bit (#095:0] get_data(); virtual function int get_data virtual function int get_dest(); vivm_fatal(get_name(), "base get_dest() called") endfunction: get_dest(); vivm_fatal(get_name(), "base get_dest() called") endfunction: get_dest(); vivtual function int get_dest(); vivtual function int get_dest(); vivtual function int get_dest(); vivtual function int get_dest(); vivtual function void my_compare(flexible_seq_item txn); vivtual function void my_compare(flexible_seq_item txn); endifunction: get_data(); 'uom_error(get_name(), "base my_compare() called") endifunction: my_compare endclass: flexible_seq_item if edit //FLEXIBLE_SEQ_ITEM_ endif //F</pre>			3	derinesroparcyvip_seq_icem				
5 uvm_sequenc_item txn_handle; 7 6 // extend the virtual function to be extended in derived 3 7 // Listing all the virtual function to be extended in derived 3 8 // UVC sequence item 11 9 virtual function new(string name); 12 11 envip.seq.item _eltem; 13 12 virtual function int get_data(); 14 12 virtual function int get_data(); 15 13 endfunction: get_data 16 14 `uvm_fatal(get_name(), "base get_data() called") 17 15 endfunction int get_addr(); 18 16 virtual function int get_addr(); 18 17 virtual function int get_addr(); 18 18 endfunction: get_data 19 19 endfunction int get_addr(); 10 10 virtual function int get_addr(); 10 11 endfunction: int get_addr(); 10 11 endfunction: int get_addr(); 10 12 virtual function int get_source(); 10 12 virtual function int get_dest();			4	<pre>class 3rdpartyvip_txn_item extends flexible_seq_item;</pre>				
<pre>6 7 // Listing all the virtual function to be extended in derived 7 // Listing all the virtual function to be extended in derived 9 virtual function new(string name); 9 virtual function new(string name); 9 virtual function new(string name); 9 virtual function int (4095:0) get_data(); 10 endfunction: new 11 endfunction: get_data 12 virtual function bit [4095:0] get_data(); 13 virtual function int get_data 14 'virtual function int get_data(); 15 endfunction: get_data 16 virtual function int get_addr(); 17 virtual function int get_addr(); 18 endfunction: get_data 16 virtual function int get_addr(); 19 endfunction: get_data 17 virtual function int get_addr(); 19 endfunction: get_data 16 virtual function int get_addr(); 19 endfunction: get_addr 20 virtual function int get_source(); 21 'vum_fatal(get_name(), "base get_source() called") 22 virtual function int get_dest(); 23 virtual function int get_dest(); 24 virtual function int get_dest(); 25 virtual function int get_dest(); 26 'virtual function int get_dest(); 27 virtual function int get_dest(); 28 virtual function int get_dest(); 29 virtual function int get_dest(); 20 'vum_fatal(get_name(), "base get_dest() called") 29 virtual function int get_dest(); 20 'vum_fatal(get_name(), "base get_dest() called") 30 endfunction: get_destination 31 endfunction: get_destination 32 endfunction: get_destination 33 virtual function void my_compare(flexible_seq_item txn); 34 'endif // _FLEXIBLE_SEQ_ITEM_ 34 'endif // _FLEXIB</pre>		<pre>class flexible_seq_item extends uvm_sequence_item;</pre>	6	<pre>// note txn_handle needs to be assigned to the extvip_seq_item</pre>				
<pre>6 7 // Listing all the virtual function to be extended in derived 7 // Listing all the virtual function to be extended in derived 9 virtual function new(string name); 9 virtual function new(string name); 10 super.new(name); 11 endfunction: new 12 virtual function bit [4095:0] get_data(); 13 virtual function bit [4095:0] get_data(); 14 virtual function bit [4095:0] get_data(); 15 endfunction: get_data 16 virtual function int get_addr(); 16 endfunction: get_data 17 virtual function int get_addr(); 18 endfunction: get_data 16 virtual function int get_addr(); 19 endfunction: get_addr 20 virtual function int get_addr(); 21 virtual function int get_addr(); 22 virtual function int get_source(); 23 virtual function int get_source(); 24 virtual function int get_dest(); 25 virtual function int get_dest(); 26 virtual function int get_dest(); 27 virtual function int get_dest(); 28 virtual function int get_dest(); 29 virtual function int get_dest(); 20 virtual function int get_dest(); 21 virtual function int get_dest(); 22 virtual function int get_dest(); 23 virtual function int get_dest(); 24 virtual function int get_dest(); 25 virtual function int get_dest(); 26 virtual function int get_dest(); 27 virtual function int get_dest(); 28 virtual function int get_dest(); 29 virtual function void my_compare(flexible_seq_item txn); 30 'urm_fatal(get_name(), "base get_dest() called") 31 endfunction: my_compare 32 endfunction: my_compare 34 'endf //FLEXIBLE_SEQ_ITEM_ 34 'en</pre>		<pre>uvm_sequence_item txn_handle;</pre>	7	// extend the virtual functions				
<pre>// Listing all the Virtual function to be extended in derived // Listing all the Virtual function to be extended in derived // UVC sequence item virtual function new(string name); virtual function inew virtual function inew virtual function inew virtual function bit [4095:0] get_data(); virtual function bit [4095:0] get_data(); virtual function int get_data virtual function int get_data virtual function int get_data virtual function int get_data virtual function int get_addr(); virtual function int get_addr virtual function int get_source(); virtual function int get_source(); virtual function int get_dest(); virtual function void my_compare(flexible_seq_item txn); virtual function void my_compare(flexible_seq_item txn); iffe_item.get_detiol is ton.get_det(); 'uwa_error(get_name(), 'wat asian iffe_item.get_detiol is ton.get_det(); 'uwa_erro</pre>	6							
<pre>// UVC sequence item // U</pre>		<pre>// Listing all the virtual function to be extended in derived</pre>	10					
<pre>9 virtual function new(string name); 10 super.new(name); 11 endfunction: new 12 i endfunction: new 13 virtual function bit [4095:0] get_data(); 14 'virtual function bit [4095:0] get_data(); 15 estile(=item,item_aliais(=_item,ids_arr); 16 eritum_serials(=_item,ids_arr); 17 virtual function int get_addr(); 18 'virtual function int get_addr(); 19 endfunction: get_data 10 virtual function int get_addr(); 10 virtual function int get_addr(); 10 virtual function int get_addr(); 11 endfunction: get_addr 12 virtual function int get_source(); 13 virtual function int get_source(); 14 virtual function int get_source(); 15 endfunction: get_datr 16 virtual function int get_source(); 17 virtual function int get_source(); 18 virtual function int get_source(); 19 endfunction: get_dest(); 10 virtual function int get_dest(); 10 virtual function int get_dest(); 10 virtual function int get_dest(); 11 virtual function void my_compare(flexible_seq_item txn); 12 virtual function: wy_compare 13 virtual function: wy_compare 14 virtual function: wy_compare 15 statice_item,intem_sitem_s</pre>				endfunction: new				
<pre>super.new(name); endfunction: new indfunction: new indfunction: new indfunction: new indfunction: new indfunction: set_idit(); intrum serialse(=[tem.data_arr]); intrum ser</pre>			13	virtual function bit [4095:0] get_data();				
<pre>endfunction: new endfunction: new endfunction: new interval i</pre>								
<pre>11 endfunction: new 12 13 virtual function bit [4095:0] get_data(); 14 `uvm_fatal(get_name(), "base get_data() called") 15 endfunction: get_data 16 17 virtual function int get_addr(); 18 `uvm_fatal(get_name(), "base get_addr() called") 19 endfunction: get_addr 20 21 virtual function int get_source(); 22 `uvm_fatal(get_name(), "base get_source() called") 23 endfunction: get_addr 24 24 25 virtual function int get_dest(); 26 `uvm_fatal(get_name(), "base get_dest() called") 27 endfunction: get_destination 28 29 virtual function void my_compare(flexible_seq_item txn); 20 20 virtual function int get_dest(); 21 intual function int get_dest(); 22 intual function int get_dest(); 23 endfunction: get_destination 24 25 virtual function int get_dest(); 26 `uvm_fatal(get_name(), "base get_dest() called") 27 endfunction: get_destination 28 29 virtual function void my_compare(flexible_seq_item txn); 20 `uvm_fatal(get_name(), "base my_compare() called") 31 endfunction: my_compare 33 `endfi // _FLEXIBLE_SEQ_ITEM 34 `endfi // _FLEXIBLE_SEQ_ITEM 35 endfunction: my_compare 36 `endfunction: my_compare 36 `endfunction: my_compare 37 endfi // _FLEXIBLE_SEQ_ITEM 35 endfunction: my_compare 36 `endfi // _FLEXIBLE_SEQ_ITEM 35 endfi // _FLEXIBLE_SEQ_</pre>								
<pre>13 virtual function bit [4095:0] get_data(); 14 `uvm_fatal(get_name(), "base get_data() called") 15 endfunction: get_data 16 'virtual function int get_addr(); 17 virtual function int get_addr(); 18 `uvm_fatal(get_name(), "base get_addr() called") 19 endfunction: get_addr 20 virtual function int get_source(); 21 virtual function int get_source(); 22 `uvm_fatal(get_name(), "base get_source() called") 23 endfunction: get_addr 24 virtual function int get_dest(); 26 `uvm_fatal(get_name(), "base get_source() called") 27 endfunction: get_destination 28 virtual function int get_dest(); 29 virtual function int get_dest(); 20 virtual function int get_dest(); 20 virtual function int get_dest(); 21 virtual function int get_dest(); 22 `uvm_fatal(get_name(), "base get_dest() called") 23 endfunction: get_destination 24 virtual function int get_dest(); 26 `uvm_fatal(get_name(), "base get_dest() called") 27 endfunction: get_destination 28 virtual function void my_compare(flexible_seq_item txn); 30 `uvm_fatal(get_name(), "base my_compare() called") 31 endfunction: my_compare 32 endclass: flexible_seq_item 33 `endif // _FLEXIBLE_SEQ_ITEM_ 34 `endif //FLEXIBLE_SEQ_ITEM_ 34 `endif //FLEXIBLE_SEQ_ITEM_ 34 `endif //FLEXIBLE_SEQ_ITEM_ 34 `endif //FLEXIBLE_SEQ_ITEM_ 34 `endif //FLEXIBLE_SEQ_ITEM_ 34 `endif //Gardentyubio Seq_item 34 `endif //FLEXIBLE_SEQ_ITEM_ 34 `endif //Gardentyubio Seq_item 34 `endif // _</pre>		endfunction: new	17					
13 Virtual function bit (##9510] get_data(); 20 extvip.seq_item e_item_int 14 `uvm_fatal(get_name(), "base get_data() called") 21 fast(citem,txn_bandle); 15 endfunction: get_data 22 return e_item_if; 16 'uvm_fatal(get_name(), "base get_addr() called") 22 virtual function int get_addr(); 16 'uvm_fatal(get_name(), "base get_addr() called") 23 fast(sitem,txn_bandle); 19 endfunction int get_source(); 'uvm_fatal(get_name(), "base get_source() called") 24 21 virtual function int get_source(); 'uvm_fatal(get_name(), "base get_source() called") 23 fast(citem,txn_handle); 22 `uvm_fatal(get_name(), "base get_dest() called") 24 fast(citem,txn_handle); 26 23 endfunction int get_dest(); 'uvm_fatal(get_name(), "base get_dest() called") 26 virtual function int get_dest(); 27 24 'uvm_fatal(get_name(), "base get_dest() called") 28 fast(e_item,txn_handle); 27 25 virtual function int get_dest(); 'uvm_fatal(get_name(), "base get_dest() called") 28 fast(e_item,txn_handle); 28 26 'uvm_fatal(get_name(), "base get_dest() called")	12							
14 `uvm_fatal(get_name(), "base get_data() called") 21 \$cast_c_item,txn_mantle); 15 endfunction: get_data 22 returm e_item.dif; 16 22 endfunction int get_addr(); 23 16 24 virtual function int get_addr(); 24 17 virtual function int get_addr() called") 26 extvip_seq_item e_item; 19 endfunction: get_addr 26 extvip_seq_item e_item; 20 'uvm_fatal(get_name(), "base get_source() called") 27 returm e_item.addr; 21 'uvm_fatal(get_name(), "base get_source() called") 28 extvip_seq_item e_item; 22 'uvm_fatal(get_name(), "base get_source() called") 28 extvip_seq_item e_item; 22 'urtual function int get_dest(); 29 virtual function int get_dest() called") 23 endfunction: get_dest 'uvm_fatal(get_name(), "base get_dest() called") 24 'uvm_fatal(get_name(), "base get_dest() called") 29 25 virtual function int get_dest(); 'urtual function in get_dest 26 'uvm_fatal(get_name(), "base my_compare() called") 20 27 endfunction: get_destination 20	13	<pre>virtual function bit [4095:0] get_data();</pre>						
<pre>endfunction: get_uata if i</pre>	14	<pre>`uvm_fatal(get_name(), "base get_data() called")</pre>	21					
16 virtual function int get_addr(); 17 virtual function int get_addr(); 18 `uvm_fatal(get_name(), "base get_addr() called") 19 endfunction: get_addr 20 'virtual function int get_source(); 21 virtual function int get_source(); 22 `uvm_fatal(get_name(), "base get_source() called") 23 endfunction: get_source 24 'virtual function int get_dest(); 25 virtual function int get_dest(); 26 'virtual function int get_dest(); 27 'scast(e_item,txn_handle); 28 endfunction: get_dest(); 29 endfunction int get_dest(); 20 'virtual function int get_dest(); 21 'virtual function int get_dest(); 22 'virtual function int get_dest(); 23 endfunction: get_destination 24 'virtual function void my_compare(flexible_seq_item txn); 29 virtual function i: get_dest 20 virtual function i: get_dest 21 endfunction: get_dest 22 virtual function void my_compare(flexible_seq_item txn); 30 'urtual fun	15	endfunction: get_data						
17virtual function int get_addr();25virtual function int get_addr();18`uvm_fatal(get_name(), "base get_addr() called")26extvip_seq_item e_item;19endfunction: get_addr2820'virtual function int get_source();28return e_item.addr;20'uvm_fatal(get_name(), "base get_source() called")31virtual function int get_source();21'uvm_fatal(get_name(), "base get_source() called")32endfunction: get_source();22`uvm_fatal(get_name(), "base get_source() called")33scast(e_item,txn_handle);23endfunction int get_dest();36endfunction int get_dest();24'urtual function int get_dest();36endfunction is get_dest();25virtual function int get_dest() called")38eatvip_seq_item e_item;26'uvm_fatal(get_name(), "base get_dest() called")38eatvip_seq_item_e_item;27endfunction: get_destination40return address_map_lookup(e_item.addr);28'urtual function void my_compare(flexible_seq_item txn);4229virtual function void my_compare(flexible_seq_item txn);4329virtual function: my_compare4420endfunction: my_compare(flexible_seq_item txn);4331endfunction: my_compare4632endfunction: my_compare4733if(e_item.get_data() != txn.get_data()) `uvm_error(get_name(), "Data mism34`endif // _FLEXIBLE_SEQ_ITEM5934`endif // _FLEXIBLE_SEQ_ITEM <t< td=""><td>16</td><td></td><td></td><td>endfunction : get_id</td></t<>	16			endfunction : get_id				
18 `uvm_fatal(get_name(), "base get_addr() called") 26 extvip_seq_item e_item; 19 endfunction: get_addr 27 return e_item.addr; 20 'uvm_fatal(get_name(), "base get_source() called") 28 return e_item.instance_no; 20 'uvm_fatal(get_name(), "base get_source() called") 31 'urtual function int get_source(); 22 `uvm_fatal(get_name(), "base get_source() called") 32 extvip_seq_item e_item; 23 endfunction int get_dest(); 32 extvip_seq_item e_item; 24 'uvm_fatal(get_name(), "base get_dest() called") 38 extvip_seq_item e_item; 26 'urtual function int get_dest(); 'urtual function int get_dest(); 'urtual function int get_dest(); 27 endfunction: get_destination 4 extvip_seq_item e_item; 'extur_seq_item.addr); 28 virtual function void my_compare(flexible_seq_item txn); 'urum_fatal(get_name(), "base my_compare() called") 38 extvip_seq_item e_item; 'extur_seq_item.addr); 29 virtual function void my_compare(flexible_seq_item txn); 'urus function void my_compare(flexible_seq_item txn); 'extur_seq_item e_item; 'scast(e_item, txn_handle); 20 endfunction: my_compare	17	virtual function int get addr():	25	virtual function int get_addr();				
<pre>19 endfunction: get_addr 20 21 virtual function int get_source(); 22 `uvm_fatal(get_name(), "base get_source() called") 23 endfunction: get_source 24 25 virtual function int get_dest(); 26 `uvm_fatal(get_name(), "base get_dest() called") 27 endfunction: get_destination 28 29 virtual function void my_compare(flexible_seq_item txn); 30 `uvm_fatal(get_name(), "base my_compare() called") 31 endfunction: my_compare 32 endclass: flexible_seq_item 33 34 `endif //FLEXIBLE_SEQ_ITEM_< 28 28 29 virtual function my_compare() 30 endfunction: my_compare 34 `endif //FLEXIBLE_SEQ_ITEM_< 28 28 29 virtual function my_compare 30 20 20 virtual function my_compare 30 31 32 34 34 `endif //FLEXIBLE_SEQ_ITEM_< 33 34 34 34 34 34 34 34 34 34 34 34 34</pre>								
<pre>19 endfunction: get_addr 20 21 virtual function int get_source(); 22 `uvm_fatal(get_name(), "base get_source() called") 23 endfunction: get_source 24 25 virtual function int get_dest(); 26 `uvm_fatal(get_name(), "base get_dest() called") 27 endfunction: get_destination 28 29 virtual function void my_compare(flexible_seq_item txn); 20 virtual function int get_dest(), "base my_compare() called") 29 endfunction: my_compare 20 virtual function int get_dest() called") 29 endfunction: my_compare 29 virtual function void my_compare(flexible_seq_item txn); 30 ` uvm_fatal(get_name(), "base my_compare() called") 31 endfunction: my_compare 32 endclass: flexible_seq_item 33 34 `endif //FLEXIBLE_SEQ_ITEM_ 30 ` indif //FLEXIBLE_SEQ_ITEM_ 31 endif //FLEXIBLE_SEQ_ITEM_ 32 endia // `FLEXIBLE_SEQ_ITEM_ 34 `endif //FLEXIBLE_SEQ_ITEM_ 35 endia // `Staget virtual set item_ 36 ` indif //FLEXIBLE_SEQ_ITEM_ 36 ` indif //FLEXIBLE_SEQ_ITEM_ 37 indif //FLEXIBLE_SEQ_ITEM_ 38 indif //FLEXIBLE_SEQ_ITEM_ 39 indif //FLEXIBLE_SEQ_ITEM_ 30 indif //FLEXIBLE_SEQ_ITEM_ 31 indif //FLEXIBLE_SEQ_ITEM_ 32 indif //FLEXIBLE_SEQ_ITEM_ 34 ` `endif //FLEXIBLE_SEQ_ITEM_ 35 indif //FLEXIBLE_SEQ_ITEM_ 36 indif //FLEXIBLE_SEQ_ITEM_ 37 indif //STOPATYVip.seq_item 39 indif //STOPATYVip.seq_item 30 indif //STOPATYVip.seq_item 31 indif //STOPATYVip.seq_item 32 indif //STOPATYVip.seq_item 33 indif //STOPATYVIP.seq_item 34 ``endif //STOPATYVIP.seq</pre>								
<pre>21 virtual function int get_source(); 22</pre>		endrunction: get_addr	29					
<pre>21 virtual function int get_source(); 22 vivm_fatal(get_name(), "base get_source() called") 23 endfunction: get_source 24 25 virtual function int get_dest(); 26 vivm_fatal(get_name(), "base get_dest() called") 27 endfunction: get_destination 28 virtual function void my_compare(flexible_seq_item txn); 30 vivm_fatal(get_name(), "base my_compare() called") 31 endfunction: my_compare 32 endclass: flexible_seq_item 33 'endif //FLEXIBLE_SEQ_ITEM_</pre> 22 extvp_seq_item e_item; 32 extvp_seq_item e_item; 32 endfunction int get_dest(); 33 extvp_seq_item e_item; 34 'endif //FLEXIBLE_SEQ_ITEM_ 32 extvp_seq_item e_item; 33 extvp_seq_item e_item; 34 extvp_seq_item e_item; 35 eadfunction int get_dest(); 36 virtual function void my_compare(flexible_seq_item txn); 37 virtual function void my_compare(flexible_seq_item txn); 38 extvp_seq_item e_item; 39 virtual function void my_compare(flexible_seq_item txn); 39 virtual function img_compare(flexible_seq_item txn); 30 is not provide the seq_item img flexible_seq_item txn); 30 is not provide the seq_item flexible_seq_item txn); 31 endif //FLEXIBLE_SEQ_ITEM_				vistual function ist act source().				
22 `uvm_fatal(get_name(), "base get_source() called") 33 \$cast(e_item,txn_handle); 23 endfunction: get_source 34 return e_item.instance_no; 24								
24 35 endfunction: get_source 25 virtual function int get_dest(); 36 26 `uvm_fatal(get_name(), "base get_dest() called") 37 27 endfunction: get_destination 38 28 virtual function void my_compare(flexible_seq_item txn); 38 29 virtual function void my_compare(flexible_seq_item txn); 41 30 `uvm_fatal(get_name(), "base my_compare() called") 43 31 endfunction: my_compare 41 32 endfunction: my_compare 46 33 `if(e_item.get_data() != txn.get_data())`uvm_error(get_name(), "Address m 34 `endif //FLEXIBLE_SEQ_ITEM 50				<pre>\$cast(e_item,txn_handle);</pre>				
<pre>24 25 virtual function int get_dest(); 26 'uvm_fatal(get_name(), "base get_dest() called") 27 endfunction: get_destination 28 29 virtual function void my_compare(flexible_seq_item txn); 30 'uvm_fatal(get_name(), "base my_compare() called") 31 endfunction: my_compare 32 endclass: flexible_seq_item 33 'endif //FLEXIBLE_SEQ_ITEM_ 36 'virtual function: my_compare 37 'utual function: my_compare 38 'endif //FLEXIBLE_SEQ_ITEM_ 30 'utual function: my_compare 34 'endif //FLEXIBLE_SEQ_ITEM_ 36 'utual function: my_compare 37 'utual function: my_compare 36 'utual function: my_compare 37 'utual function: my_compare 38 'endif //FLEXIBLE_SEQ_ITEM_ 39 'utual function: my_compare 30 'utual function: my_compare 31 'endif //FLEXIBLE_SEQ_ITEM_ 32 'utual function: my_compare 34 'endif //FLEXIBLE_SEQ_ITEM_ 35 'utual function: my_compare 36 'utual function: my_compare 36 'utual function: my_compare 36 'utual function: my_compare 37 'utual function: my_compare 38 'endif //FLEXIBLE_SEQ_ITEM_ 39 'utual function: my_compare 30 'utual function: my_compare 31 'utual function: my_compare 32 'utual function: my_compare 33 'utual function: my_compare (my_compare(), "Address my_compare(), "Address my_compare(), "atdpartytyip_seq_item</pre>		endfunction: get_source						
26 `uvm_fatal(get_name(), "base get_dest() called") 38 extvip_seq_item e_item; 26 `uvm_fatal(get_name(), "base get_dest() called") 39 scast(e_item,txn_handle); 27 endfunction: get_destination 41 endfunction : get_destination 28 virtual function void my_compare(flexible_seq_item txn); 41 endfunction : get_dest 29 virtual function void my_compare(flexible_seq_item txn); 42 30 `uvm_fatal(get_name(), "base my_compare() called") 44 extvip_seq_item e_item; 30 `uvm_fatal(get_name(), "base my_compare() called") 45 scast(e_item, txn_handle); 31 endfunction: my_compare 45 scast(e_item, get_data()) `uvm_error(get_name(), "Data mism 33 `endif //FLEXIBLE_SEQ_ITEM 50 endfunction: my_compare 34 `endif //FLEXIBLE_SEQ_ITEM 50 endclass: 3rdpartyvip_seq_item	24							
26 'uvm_fatal(get_name(), "base get_dest() Called") 39 \$cast(e_item, txn_handle); return address_map_lookup(e_item.addr); 27 endfunction: get_destination 40 return address_map_lookup(e_item.addr); 28 'uvm_fatal(get_name(), "base my_compare(flexible_seq_item txn); 40 endfunction void my_compare(flexible_seq_item txn); 29 virtual function void my_compare(flexible_seq_item txn); 41 endfunction: void my_compare(flexible_seq_item txn); 30 `uvm_fatal(get_name(), "base my_compare() called") 43 virtual function void my_compare(flexible_seq_item txn); 31 endfunction: my_compare 46 if(e_item.get_data()) !uvm_error(get_name(), "Data mism 33 'endif //FLEXIBLE_SEQ_ITEM 50 endclass: 3rdpartyvip_seq_item 34 `endif //FLEXIBLE_SEQ_ITEM 50 endclass: 3rdpartyvip_seq_item	25	<pre>virtual function int get_dest();</pre>						
27 endfunction: get_destination 40 return address_map_lookup(e_item.addr); 28	26	<pre>`uvm_fatal(get_name(), "base get_dest() called")</pre>						
28 42 29 virtual function void my_compare(flexible_seq_item txn); 30 `uvm_fatal(get_name(), "base my_compare() called") 31 endfunction: my_compare 32 endfunction: my_compare 33 `endif //FLEXIBLE_SEQ_ITEM 34 `endif //FLEXIBLE_SEQ_ITEM	27	endfunction: get_destination	40					
<pre>29 virtual function void my_compare(flexible_seq_item txn); 30 `uvm_fatal(get_name(), "base my_compare() called") 31 endfunction: my_compare 32 endclass: flexible_seq_item 33 34 `endif //FLEXIBLE_SEQ_ITEM</pre>	28			endfunction : get_dest				
30 `uvm_fatal(get_name(), "base my_compare() called") 44 extvip.seq_item e_item; 31 endfunction: my_compare 45 \$cast(e_item, txn_handle); 32 endclass: flexible_seq_item 46 33 if(e_item.get_data()) 'uvm_error(get_name(), "Data mism 34 `endif //FLEXIBLE_SEQ_ITEM 50 34 `endif //FLEXIBLE_SEQ_ITEM 50		virtual function void my compare(flexible seg item typ):		virtual function void my compare(flexible seg item txn):				
31 endfunction: my_compare 43 stast(E_item, txn_nand(e); 32 endclass: flexible_seq_item 47 if(e_item.get_ada()) = txn.get_ada())`uvm_error(get_name(), "Data mism 33 33 34 `endif //FLEXIBLE_SEQ_ITEM 50 endclass: 3rdpartyvip_seq_item 34 `endif //FLEXIBLE_SEQ_ITEM 50 endclass: 3rdpartyvip_seq_item 51			44	<pre>extvip_seq_item e_item;</pre>				
32 endclass: flexible_seq_item 47 if(e_item.get_data() = txn.get_data()) `uvm_error(get_name(), "Data mism 33 38 48 if(e_item.get_addr()) 'uvm_error(get_name(), "Address m 34 `endif //FLEXIBLE_SEQ_ITEM 50 endclass: 3rdpartyvip_seq_item				<pre>\$cast(e_item, txn_handle);</pre>				
32 endticass: flexible_seq_item 48 if(e_item.get_addr() != txn.get_addr()) `uvm_error(get_name(), "Address m 33 49 endfunction: my_compare 34 `endif //FLEXIBLE_SEQ_ITEM 50 51 `endif //Srdpartyvip_seq_item				<pre>if(e_item.get_data() != txn.get_data()) `uvm_error(get_name(), "Data mismatched!")</pre>				
34 `endif //FLEXIBLE_SEQ_ITEM50 endclass: 3rdpartyvip_seq_item		endclass: flexible_seq_item	48	<pre>if(e_item.get_addr() != txn.get_addr()) `uvm_error(get_name(), "Address mismatched!"</pre>				
34 endit //FLEXIBLE_SEQ_IIEM								
		endit //FLEXIBLE_SEQ_ITEM						
52	35		52					



V. PROXY MONITOR CLASSES

Flexible Checker is not just a scoreboard but a combination of a scoreboard and the corresponding monitor feeding it. The monitors could be either monitors for protocols generated in house or from a third party. These monitors send transaction of the container transaction item type via the analysis ports. The idea behind instantiating monitors alongside the scoreboard is to bypass the process of instantiating an agent in passive mode and to add it to the env instead. This makes it simple to instantiate the monitor and set the corresponding configuration database in one place. Also, this gives the flexibility to change the number and type of instantiations. Reference code is shown below to support our in-house and third-party VIPs.

	inhouse_monitor.sv		3rdPartyVIP_Monitor.sv	
1	`ifndefINHOUSE_MONITOR	1	`ifndef3rdparty_monitor	
2	<pre>`defineINHOUSE_MONITOR</pre>	2	`define3rdparty_monitor	
3		3	<pre>class 3rdparty_monitor extends flexible_monitor;</pre>	
4	<pre>class inhouse_monitor extends flexible_monitor;</pre>	4	<pre>// This port receives the expvip_seq_item and puts it</pre>	
5	<pre>function new(string name, uvm_component parent);</pre>	5	<pre>// in the container class derived from flexible_seq_item</pre>	
6	<pre>super.new(name, parent);</pre>	6	uvm_analysis_import #(extvip_seq_item) imp_port;	
7	endfunction : new	7	extvip_monitor monitor;	
8		8		
9	task run;	9 10	<pre>function new(string name, uvm_component parent); imp_port = new("imp_port", this);</pre>	
10	<pre>flexible_seq_item base_txn;</pre>	11	<pre>mmp_port = new(imp_port ; this); monitor = extvip_monitor::type_id::create("monitor", this);</pre>	
11	inhousevip seg item txn;	12	endfunction: new	
12	· · · · · · · · · · · · · · · · · · ·	13		
13	forever begin	14	<pre>function write(extvip_seg_item txn);</pre>	
14	@(cb.clk);	15	<pre>3rdpartyvip_seq_item cntr_txn; // extended from flexible_seq_item</pre>	
15		16	<pre>flexible_seq_item send_txn;</pre>	
16	<pre>// create a transaction and assign the signals</pre>	17		
17	<pre>txn = inhousevip_seq_item::type_id::create("txn");</pre>	18	<pre>cntr_txn = 3rdparty_seq_item::type_id::create("cntr_txn");</pre>	
18	txn.addr = cb.addr;	19	<pre>cntr_txn.txn_handle = txn;</pre>	
19	txn.data = cb.data;	20	<pre>send_txn = cntr_txn;</pre>	
20		21 22	analysis_port.write(send_txn);	
21	<pre>// create the flexible transactions</pre>	22	endfunction: write	
22	<pre>base_txn = txn;</pre>	24		
23	analysis_port.write(base_txn);	25	<pre>function connect_phase(uvm_phase phase);</pre>	
24	end	26	<pre>monitor.analysis_port.connect(this.imp_port);</pre>	
25	endtask: run	27	endfunction: connect_phase	
26	endclass: inhouse monitor	28		
27	`endif //INHOUSE_MONITOR	29	endclass: 3rdparty_monitor	
28		30	<pre>`endif //3rdparty_monitor</pre>	



VI. ELASTIC-/-FLEXIBLE SCOREBOARD

The crux of the scoreboard is its ability to add expected queues for each of the end nodes or targets. The idea is that whenever a transaction is reported at a source, it has inherent knowledge of which destination it must go to, or this information is provided to the scoreboard beforehand via a config item, then it pushes the transaction into the corresponding expected queue. Also, the transactions received by the scoreboard are casted to a base class type called the proxy transaction class, which enables the scoreboard to operate on a single transaction type. Once a transaction is reported by the target, it compares the transaction with the one in its corresponding expected queue. At the end of the simulation, the queues are checked to be empty, or the test fails and reports the stray transactions.

Flexible_Scoreboard.sv	Flexible_Scoreboard.sv
1 `ifndefFLEXIBLE_SCOREBOARD	
2 `defineFLEXIBLE_SCOREBOARD	59 end
3	<pre>60 foreach(m1_array_of_queues_out[i]) begin 61 if(m1 array of queues out[i].size()) begin</pre>
4 `uvm_analysis_imp_decl(_m0_in_0)	61 if(m1_array_of_queues_out[i].size()) begin 62 `uvm_error(get_name(), \$psprintf("m1_array_of_queues_out[%0d] is not empty [size:%0d]",
5 `uvm_analysis_imp_decl(_m0_in_1)	<pre>63 0vm_error(get_name(), spsprint((mi_array_of_queues_out[wdg] is not empty [size: dog] , 63</pre> i, m1_array_of_queues_out[i].size()))
6 `uvm_analysis_imp_decl(_m1_in_0)	64 end
7 `uvm_analysis_imp_decl(_m1_in_1)	65 end
<pre>8 `uvm_analysis_imp_decl(_m0_out_0)</pre>	66 endfunction: check_phase
<pre>9 `uvm_analysis_imp_decl(_m0_out_1)</pre>	67
<pre>10 `uvm_analysis_imp_decl(_m1_out_0)</pre>	<pre>68 virtual function void write_m0_in_0(flexible_seq_item txn);</pre>
<pre>11 `uvm_analysis_imp_decl(_m1_out_1)</pre>	<pre>69 this.register_m0_input(txn);</pre>
12 13 // typedef for queues	70 endfunction: write_m0_in_0
<pre>13 77 typedef for queues 14 typedef flexible_seq_item flex_item_queue[\$];</pre>	71
14 cypeder reexible_sed_rem reex_rem_dede(a);	72 virtual function void write_m0_in_1(flexible_seq_item txn);
16 virtual class flexible_scoreboard extends uvm_scoreboard;	<pre>73 this.register_m0_input(txn);</pre>
<pre>17 `uvm_component_utils(flexible_scoreboard)</pre>	74 endfunction: write_m0_in_1
18	75
<pre>19 flexible_item_queue m0_array_of_queues_out[];</pre>	<pre>76 virtual function void write_m1_in_0(flexible_seq_item txn);</pre>
<pre>20 flexible_item_queue m1_array_of_queues_out[];</pre>	77 this.register_m1_input(txn);
21	78 endfunction: write_m1_in_0 79
22 uvm_analysis_imp_m0_in_0 #(flexible_seq_item, flexible_scoreboard) m0_imp_in_0;	<pre>//9 80 virtual function void write_m1_in_1(flexible_seq_item txn);</pre>
23 uvm_analysis_imp_m0_in_1 #(flexible_seq_item, flexible_scoreboard) m0_imp_in_1;	<pre>80 Virtual function void write_mi_in_if(fexible_sed_ifem (xn); 81 this.register_m1_input(txn);</pre>
<pre>24 uvm_analysis_imp_m1_in_0 #(flexible_seq_item, flexible_scoreboard) m1_imp_in_0;</pre>	82 endfunction: write_m1_in_1
<pre>25 uvm_analysis_imp_m1_in_1 #(flexible_seq_item, flexible_scoreboard) m1_imp_in_1;</pre>	sa
26	<pre>84 virtual local function void register_m0_input(flexible_seq_item txn);</pre>
<pre>27 uvm_analysis_imp_m0_out_0 #(flexible_seq_item, flexible_scoreboard) m0_imp_out_0;</pre>	<pre>85 m@_array_of_queues_out[txn.get_dest()].push_back(txn);</pre>
28 uvm_analysis_imp_m0_out_0 #(flexible_seq_item, flexible_scoreboard) m0_imp_out_1; 29 uvm analysis imp m1 out 0 #(flexible seq item, flexible scoreboard) m1 imp out 0:	86 endfunction: register_m0_input
29 uvm_analysis_imp_m1_out_0 #(flexible_seq_item, flexible_scoreboard) m1_imp_out_0; 30 uvm_analysis_imp_m1_out_0 #(flexible_seq_item, flexible_scoreboard) m1_imp_out_1;	87
31	<pre>88 virtual local function void register_m1_input(flexible_seq_item txn);</pre>
<pre>32 function new(string name, uvm_component parent);</pre>	<pre>89 m1_array_of_queues_out[txn.get_dest()].push_back(txn);</pre>
33 super.new(name, parent);	90 endfunction: register_m1_input
34 endfunction: new	91
35	<pre>92 virtual function void write_m0_out_0(flexible_seq_item txn);</pre>
<pre>36 function build_phase(uvm_phase phase);</pre>	93 this.check_m0_output(txn); 94 endfunction: write m0 out 0
<pre>37 m0_imp_in_0 = new("m0_imp_in_0", this);</pre>	94 endfunction: write_m0_out_0 95
<pre>38 m0_imp_in_1 = new("m0_imp_in_1", this);</pre>	<pre>96 virtual function void write_m0_out_1(flexible_seq_item txn);</pre>
<pre>39 m1_imp_in_0 = new("m1_imp_in_0", this);</pre>	97 this.check.me_output(txn);
<pre>40 m1_imp_in_1 = new("m1_imp_in_1", this);</pre>	98 endfunction: write_m0_out_1
<pre>41 m0_imp_out_0 = new("m0_imp_out_0", this);</pre>	99
<pre>42 m0_imp_out_1 = new("m0_imp_out_1", this); 43 m1 imp out 0 = new("m1 imp out 0", this);</pre>	<pre>100 virtual function void write_m1_out_0(flexible_seq_item txn);</pre>
	101 this.check_m1_output(txn);
<pre>44 m1_imp_out_1 = new("m1_imp_out_1", this); 45</pre>	<pre>102 endfunction: write_m1_out_0</pre>
45 m0_array_of_queues_out = new[2];	103
47 m1_array_of_queues_out = new[2];	<pre>104 virtual function void write_m1_out_1(flexible_seq_item txn);</pre>
48	<pre>105 this.check_m1_output(txn);</pre>
49 endfunction: build_phase	106 endfunction: write_m1_out_1
50	<pre>107 108 virtual local function void check_m0_output(flexible_seq_item txn);</pre>
<pre>51 virtual function void check_phase(uvm_phase phase);</pre>	<pre>106 Virtual total function void check_mo_output(flexible_sed_item txn); 109 flexible_sed_item exp_txn;</pre>
52 super.check_phase(phase);	<pre>109</pre>
53	111 txn.my_compare(exp_txn);
54 foreach(m0_array_of_queues_out[i]) begin	112 endfunction: check.m0.output
<pre>55 if(m0_array_of_queues_out[i].size()) begin</pre>	445



VII. FLEXIBLE CHECKER

Flexible Checker gives the capability to check a transaction flowing through different nodes in a SoC. When a transaction is reported at a source, the checker populates the expected queues of all the internal nodes the transactions is expected to go to. Once a transaction is reported at these internal nodes, they are checked across the elements in their corresponding expected queues (which were populated by the source). While flowing through nodes, the transaction translates to different types/protocols, which requires coders to write a translation function that can then be reused across the scoreboard when such a translation is expected.

	Flexible_Checker.sv
1	`ifndefFLEXIBLE_CHECKER
2	`defineFLEXIBLE_CHECKER
3	
4	virtual class flexible_checker extends uvm_component;
5	<pre>`uvm_component_utils(flexible_checker)</pre>
6	
7	<pre>// derived from the datastructure defined in our database of topology</pre>
8	<pre>inhouse_monitor m0_monitors_in0;</pre>
9	<pre>inhouse_monitor m0_monitors_in1;</pre>
10	<pre>3rdparty_monitor m1_monitors_in0;</pre>
11	<pre>3rdparty_monitor m1_monitors_in1;</pre>
12	
13	inhouse_monitor m0_monitors_out0;
14	<pre>inhouse_monitor m0_monitors_out1;</pre>
15	3rdparty_monitor m1_monitors_out0;
16	3rdparty_monitor m1_monitors_out1;
17	
18	flexible_scoreboard m_scoreboard;
19	
20	<pre>function new(string name, uvm_component parent);</pre>
21	<pre>super.new(name, parent);</pre>
22	endfunction: new
23	
24	<pre>virtual function void build_phase(uvm_phase phase);</pre>
25	<pre>super.build_phase(phase);</pre>
26	
27	// building of required monitors
28	<pre>m0_monitors_in0 = inhouse_monitor::type_id::create("m0_monitors_in0", this);</pre>
29	<pre>m0_monitors_in1 = inhouse_monitor::type_id::create("m0_monitors_in1", this);</pre>
30	<pre>m1_monitors_in0 = 3rdparty_monitor::type_id::create("m1_monitors_in0", this); and analytic inf = 2rdparty_monitor::type_id::create("m1_monitors_in1", this);</pre>
31	<pre>m1_monitors_in1 = 3rdparty_monitor::type_id::create("m1_monitors_in1", this);</pre>
32 33	π^0 manifers out 0 - inhouse manifestations iduces $(\pi^0 \pi^0 \pi^0 ^2)$
34	<pre>m0_monitors_out0 = inhouse_monitor::type_id::create("m0_monitors_out0", this); m0_monitors_out1 = inhouse_monitor::type_id::create("m0_monitors_out1", this);</pre>
35	<pre>mme_monitors_out1 = innouse_monitor::type_id::create("m1_monitors_out0", this); m1_monitors_out0 = 3rdparty_monitor::type_id::create("m1_monitors_out0", this);</pre>
36	<pre>m1_monitors_out0 = 3rdparty_monitor::type_id::create(m1_monitors_out0 ; this); m1_monitors_out1 = 3rdparty_monitor::type_id::create("m1_monitors_out1", this);</pre>
37	endfunction: build_phase
38	charanterion. burta_phase
39	<pre>virtual function void connect_phase(uvm_phase phase);</pre>
40	<pre>super.connect_phase(phase);</pre>
41	
42	<pre>// connection of monitors to the respective ports in</pre>
43	// scoreboard
44	<pre>m0_monitors_in0.analysis_port.connect(m_scoreboard.m0_imp_in_0);</pre>
45	<pre>m0_monitors_in1.analysis_port.connect(m_scoreboard.m0_imp_in_1);</pre>
46	<pre>m1_monitors_in0.analysis_port.connect(m_scoreboard.m1_imp_in_0);</pre>
47	<pre>m1_monitors_in1.analysis_port.connect(m_scoreboard.m1_imp_in_1);</pre>
48	
49	<pre>m0_monitors_out0.analysis_port.connect(m_scoreboard.m0_imp_out_0);</pre>
50	<pre>m0_monitors_out1.analysis_port.connect(m_scoreboard.m0_imp_out_1);</pre>
51	<pre>m1_monitors_out0.analysis_port.connect(m_scoreboard.m1_imp_out_0);</pre>
52	<pre>m1_monitors_out1.analysis_port.connect(m_scoreboard.m1_imp_out_1);</pre>
53	endfunction: connect_phase
54	endclass: flexible_checker
55	

VIII. TRANSACTION TRACING

All transaction tracing is done using Flexible Checker, which includes

- 1. end-to-end checking to make transactions reach the correct port, in the correct order, and with the correct content,
- 2. selectively monitor any/all interfaces of an interconnect to trace the transaction route (or path) for correctness and flag an error otherwise,
- 3. selectively trace any/all transactions (using a preconfigured transaction ID) across an interconnect,



- 4. track timeout transactions (transactions that took more time than maximum allocated time in cycles),
- 5. track dropped transactions,
- 6. track split and merged transactions, and
- 7. track multicast and broadcast transactions.

IX. STATISTIC MONITORING & DUMPING

Flexible Checker can also generate statistics, including latency, bandwidth, traffic distribution, and histogram data, with the transactions reported at the source and the destination and the relationship between them established by the scoreboard. The corresponding timing and delay information is stored on a master/ slave basis in the scoreboard and is then used to create textual or graphical data.

X. CONCLUSION

Flexible Checker was used in a highly configurable test bench and was able to do the checker functions just by passing the configuration parameters. It made building a block-level test bench redundant by using the transaction trace-checking feature, which helped the designer greatly in debugging failures as they were able to see the transaction flow and translation at each node of the design before reaching the target. Additionally, a statistics generator was used to print a table listing the transaction count, latency, and bandwidth numbers. This gave firsthand statistic numbers to the architects and gave a comparison point with the actual performance model numbers.